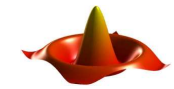
SIMULINK

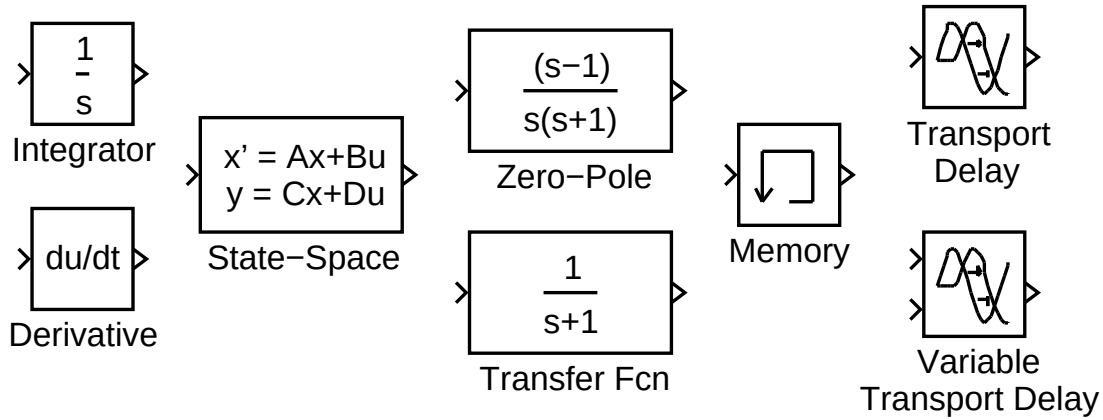
Linear and non-linear

Systems



SIMULINK Linear & Non-Linear Systems
SIMULINK Library Continuous

- Elements for modelling continuous-time systems
- Delays



SIMULINK Linear & Non-Linear Systems
From ODE to Continuous–Block

Example: spring pendulum

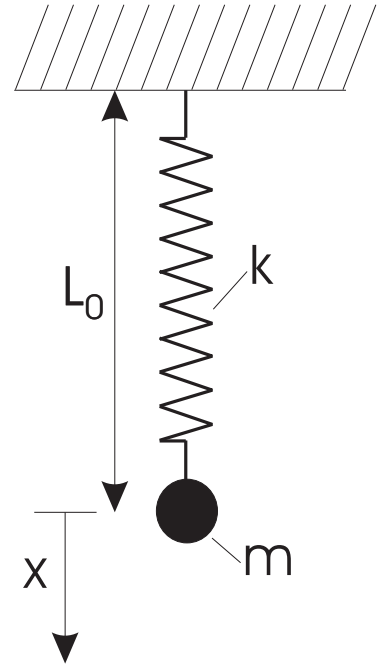
ODE: $m \cdot \ddot{x}(t) + k \cdot x(t) = 0$

Solution in time domain:

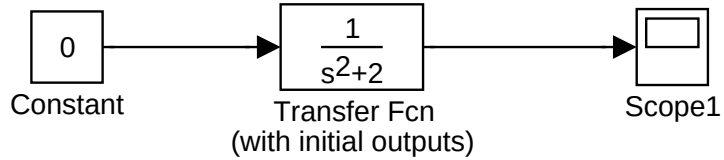
$x(t) = x_0 \cos(\omega_0 t) + \frac{v_0}{\omega_0} \sin(\omega_0 t)$

Transform to frequency domain:

Transform function $X(s) = \frac{1}{s^2 + \omega_0^2}$

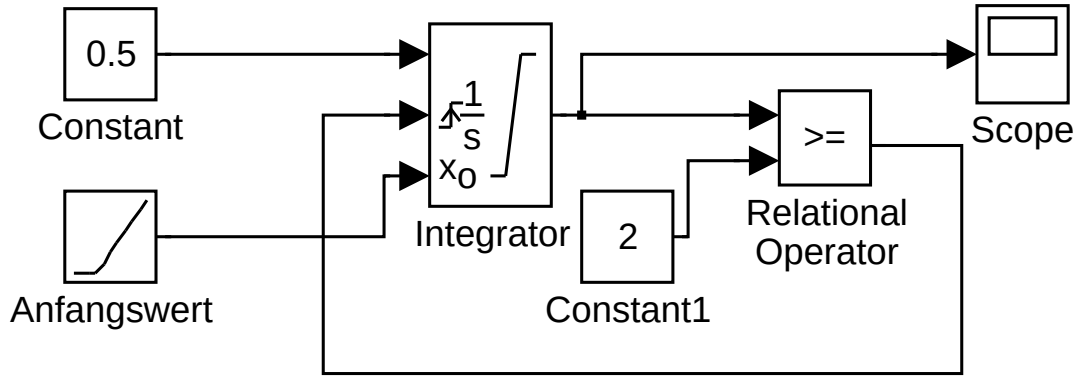


$\Rightarrow$ in SIMULINK:



SIMULINK Linear & Non-Linear Systems
Examples for Continuous

Resetting of Simulink *Integrator* Block:



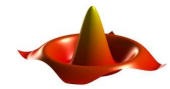
SIMULINK Linear & Non-Linear Systems
Linearisation

- Linearisation of simulation models at arbitrary operating point
- Command:

  [A,B,C,D] = `linmod('sys',`x`,`u`,`$para$`,`$xpert$`,`$upert$`)`

  [Ad,Bd,Cd,Dd] = `dlinmod('sys',`Ts`,`x`,`u`)`

- Further Matlab–Commands from Control System Toolbox for analysing (`ss`, `tf`, `zpk`, `bode`, `lsim`)

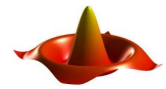


SIMULINK Linear & Non-Linear Systems
Find a point of equilibrium

- A trim point (point of equilibrium) corresponds to the steady state of a dynamic system

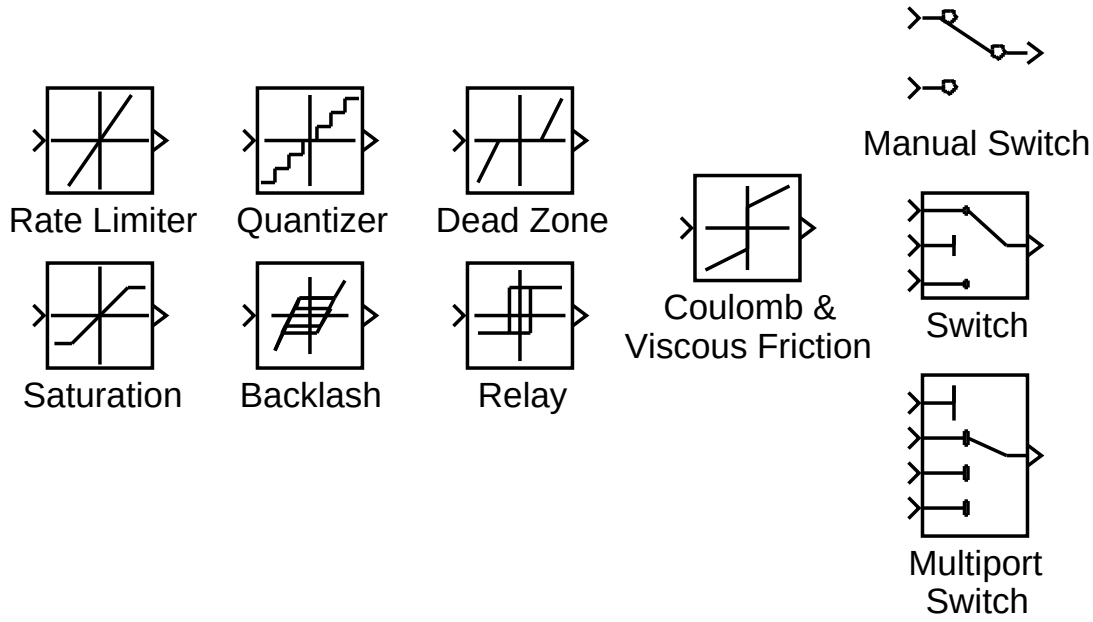
$$\dot{\mathbf{x}} = \mathbf{0}$$

- Command: `[`x`,`u`,`y`] = trim('`sys`', `$x0$`, `$u0$`, `$y0$`)`
- further options $\Rightarrow$ `help trim`



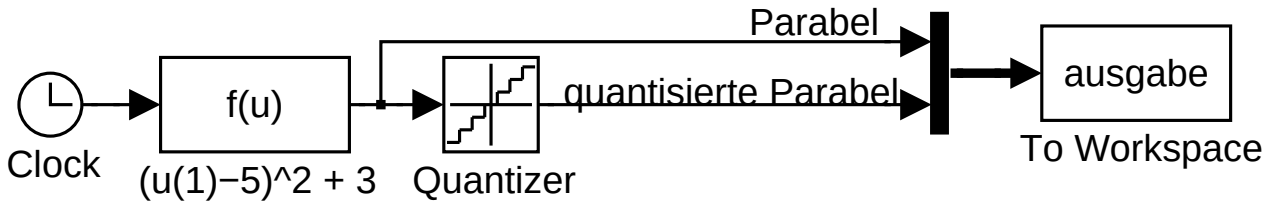
SIMULINK Linear & Non-Linear Systems
SIMULINK Library Nonlinear

- Elements for modelling of physical non-linearities
- Switches

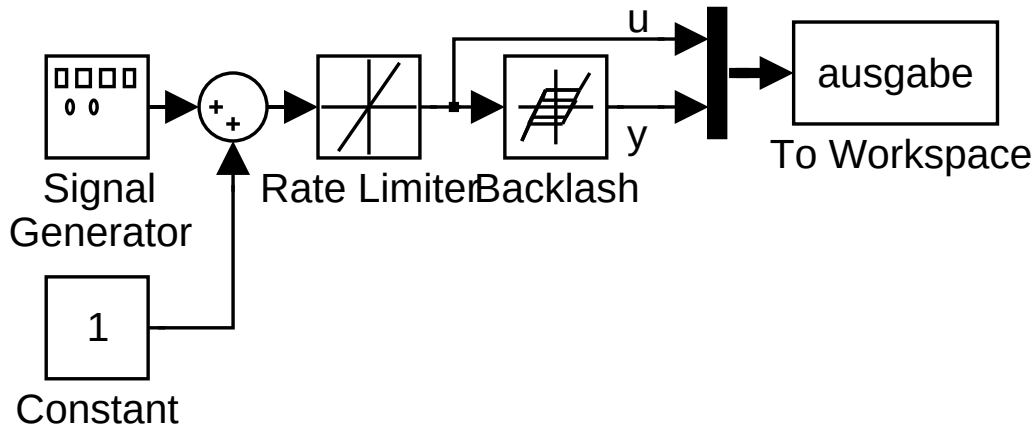


SIMULINK Linear & Non-Linear Systems
Example for Nonlinear

Example for *Quantizer*

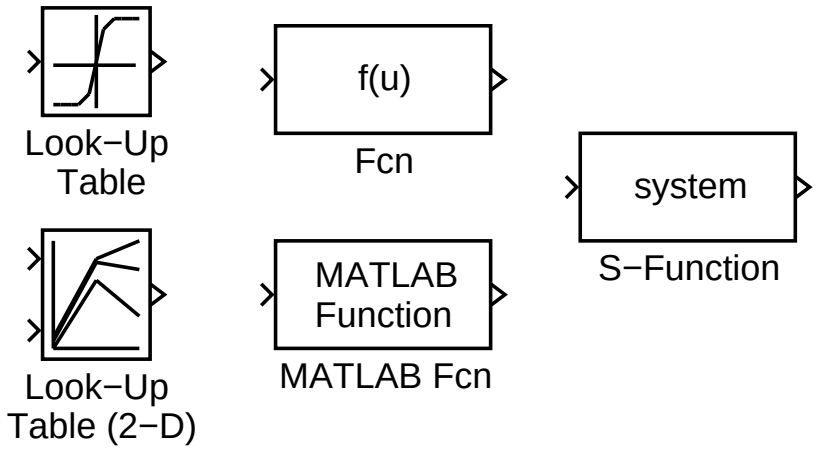


Example for *Backlash*



SIMULINK Linear & Non-Linear Systems
SIMULINK Library Functions & Tables

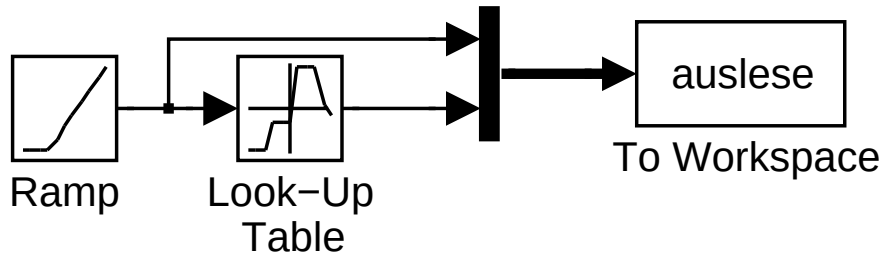
- Tables
- Programmable functions



etc.

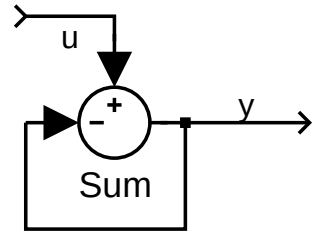
SIMULINK Linear & Non-Linear Systems
Examples for Function & Tables

Examples for *Look–Up Table*



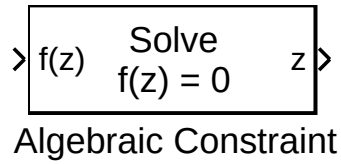
SIMULINK Linear & Non-Linear Systems
Algebraic loops

- *direct feedthrough*: Output port of a block drives input port of the same block, i.e. input depends on output at the same time



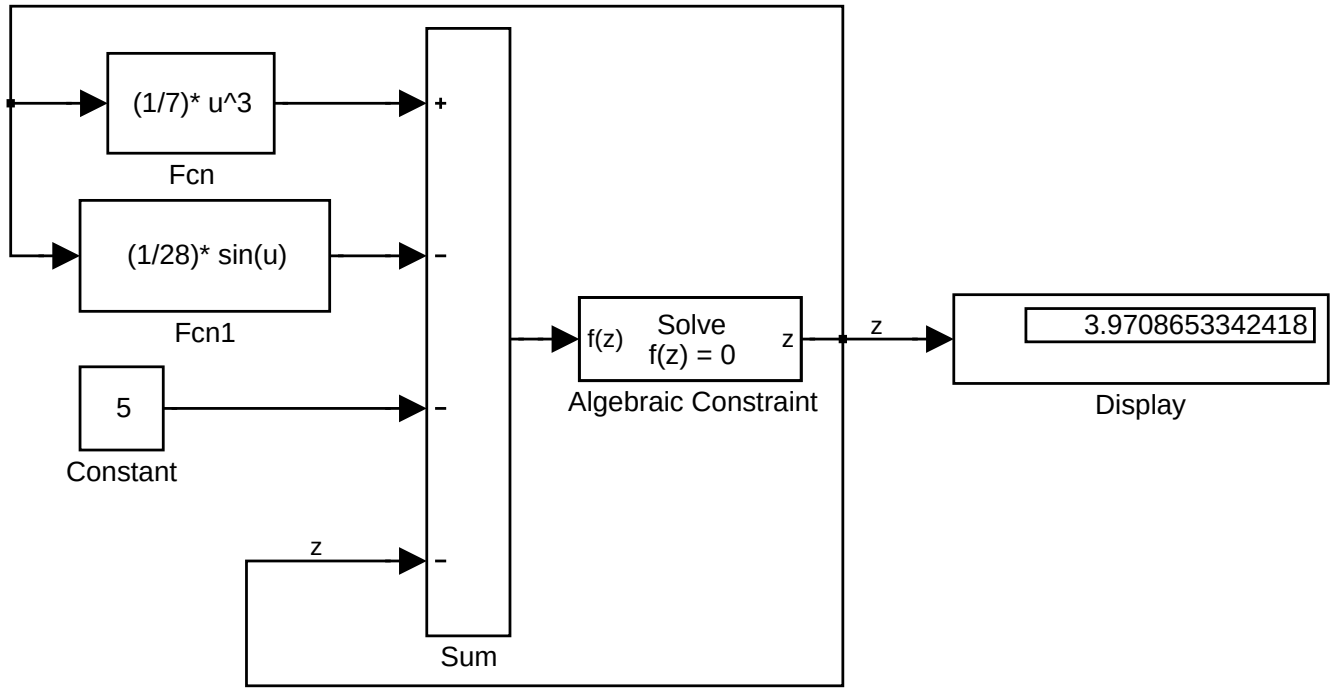
- Blocks with *direct feedthrough*: *Sum*, *Gain*, *Product* (*State Space*, *Integrator*, *Transfer Function*, *Zero–Pole*)

- Solution with *Algebraic Constraint*:



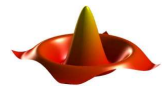
SIMULINK Linear & Non-Linear Systems
Example for algebraic loops

$$f(z) = \frac{1}{7}z^3 - \frac{1}{28}\sin z - z - 5 = 0$$



SIMULINK Linear & Non-Linear Systems
S–Functions

- Self-programmable functions
- Use *S–Function*–Block in SIMULINK–Model
- programmable in
  - Matlab (M–Files)
  - C, C++, Fortran, Ada (MEX Files)
- Integration of earlier written programmes

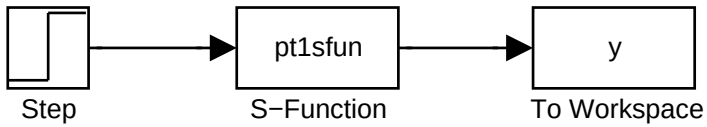


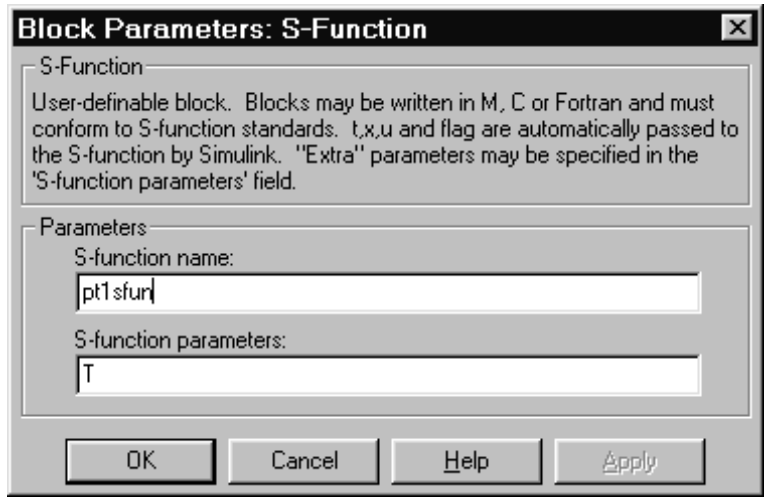
SIMULINK Linear & Non-Linear Systems
Example for S–Functions

PT_1:

Transfer function: $$Y(s) = \frac{1}{1+sT} \cdot U(s)$$

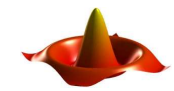
or state space: $$\dot{x} = -\frac{1}{T} \cdot x + \frac{1}{T} \cdot u$$
$$y = x$$





SIMULINK Linear & Non-Linear Systems
Example for S–Functions

```
function [sys,x0,str,ts] = pt1sfun(t,x,u,flag,T)

A = [-1/T];     % State matrix
B = [ 1/T];     % Input matrix
C = [ 1  ];     % Output matrix
D = [ 0  ];     % Feedthrough matrix=0
switch flag,

% Initialization %
case 0,
  [sys,x0,str,ts]=mdlInitializeSizes(A,B,C,D);

% Derivatives %
case 1,
  sys=mdlDerivatives(t,x,u,A,B,C,D);

% Outputs %
case 3,
  sys=mdlOutputs(t,x,u,A,B,C,D);

% not used flags
case {2, 4, 9},
  sys=[];

% Unexpected flags %
otherwise          % error handling
  error(['Unhandled flag = ',num2str(flag)]);
end
% end pt1sfun


%====================
% mdlInitializeSizes
%====================
%
function [sys,x0,str,ts]=...
     mdlInitializeSizes(A,B,C,D)

sizes = simsizes;

sizes.NumContStates  = size(A,1);
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = size(C,1);

sizes.NumInputs      = size(B,1);
```

SIMULINK Linear & Non-Linear Systems
Example for S–Functions

```
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;

sys = simsizes(sizes);

x0  = zeros(size(A,1),1);

str = [];

ts  = [0 0];
% end mdlInitializeSizes

%==================
% mdlDerivatives
%==================
%
function sys=mdlDerivatives(t,x,u,A,B,C,D)

sys = A*x + B*u;

% end mdlDerivatives
```

```
%====================
% mdlOutputs
%====================
%
function sys=mdlOutputs(t,x,u,A,B,C,D)

sys = C*x + D*u;

% end mdlOutputs
```